Dados do projeto

Nome do Projeto:	*API Card Reader*		
Responsável:	*Ronaldo Bettiol*		
Data da Abertura:		**Client:**	*Perto*

Contents

1. Introduction

This API is provided as a multiplatform dynamic library.

2. This documentation provides information for developer to integrate the application with card reader library. All functions are C style and are described in the Compatibility

This API is provided as a multiplatform dynamic library which supports the following system:

- *Windows 32/64 bits*
- *Linux 32/64 bits*

Exported Function section.

First function application need call is the IDC_Open() to create a new instance of communication with device, it's necessary pass the service name like was described in the configuration file, so IDC_Open()returns a index. This index will be used for any function.

3. Compatibility

This API is provided as a multiplatform dynamic library which supports the following system:

- *Windows 32/64 bits*
- *Linux 32/64 bits*

4. Exported Function

4.1.IDC_Open

```
int IDC_Open(char *service, int *idx);
```

*This function open the communication port to access the device, the parameter service is used to find the section in configuration file with the same name and read all configuration necessary to create an internal service into PertoCardReader library. The second parameter (*idx) is a pointer for the service index, it will be used in all posterior function calls.*

Parameters

Input:

`char *service`: name of the service will be create, same name of section in configuration file, necessary same value in IDC_Close.

Output:

`int *idx`: pointer to receive the index of service created, necessary in all posterior function calls.

Example Code:

```c
int ret = 0;
int idx = 0; //necessary in all other functions
char service[] = "SERVICE0"; //necessary in IDC_Close
ret = IDC_Open(service, &idx);
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.2.IDC_Close

`int IDC_Close(char *service, int *idx);`

This function close the communication port and delete the service created in IDC_Open.

Parameters

Input:

*char *service*: same used in IDC_Open.

*int *idx*: pointer to service index created in IDC_Open.

Output:

*int *idx*: if success the value will be changed to -1

Example:

```c
int ret = 0;
int idx = 0; //same used in PIN_Open
char service[] = "SERVICE0"; //same used in PIN_Open

ret = IDC_Close(service, &idx);
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.3.IDC_Reset

`int IDC_Reset(int idx);`

This function performs a software reset on device.

Parameters

Input:

int idx: Same index returned by IDC_Open.

Example:

```c
int ret = 0;

ret = IDC_Reset(idx);
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.4.IDC_GetDeviceInfo

```
int IDC_GetDeviceInfo(int idx,
                    char            *szModel,
                    char            *szEPROMVersion,
                    char            *szIOResource,
                    unsigned int    *dwDevVersion,
                    unsigned short  *wType,
                    unsigned int    *dwCaps,
                    unsigned short  *wChipProtocol,
                    unsigned int    *dwMaxCards);
```

This function close the communication port and delete the service created in PIN_Open.

Parameters

Input:

int idx: Same index returned by IDC_Open.

Output:

*char *szModel*: Returns device model as a null terminated string.

*char *szEPROMVersion*: Returns device revision as a null terminated string.

*char *szIOResource*: Returns the communication settings as a null terminated string.

*unsigned int *dwDevVersion*: Device revision.

*unsigned short *wType*: Returns device type. Possible values are:

```
0x0001 – Device has a motor
0x0002 – Device is sliding
0x0004 – Device
0x0008 – Contact less.
0x0010 – Device has SAM module
```

*unsigned int *dwCaps*: Returns the device’s features. Possible values are.

```
0x00000001 – Device is a device component
0x00000002 – Device has support to eject
0x00000004 – Device has support to capture cards
0x00000008 – Device has support to read magnetic track 1
0x00000010 – Device has support to read magnetic track 2
0x00000020 – Device has support to read magnetic track 3
0x00000040 – Device has support to write magnetic track 1
0x00000080 – Device has support to write magnetic track 2
0x00000100 – Device has support to write magnetic track 3
0x00000200 – Device has support to smart cards
0x00000400 – Device has support to cryptography
0x00000800 – Device has support to magnetic
0x00001000 – Device has support to leds
```

*unsigned short *wChipProtocol*: Indicate smart card protocol when available. Possible values are 0-15 meaning T0-T15.

*unsigned int *dwMaxCards*: Maximum number of retained cards.

Example Code:

```
int ret = 0;
char Model[64];
char EPROMVersion[64];
char IOResource[64];
unsigned int DevVersion;
unsigned short Type;
unsigned int Caps;
unsigned short ChipProtocol;
unsigned int MaxCards;

ret = IDC_GetDeviceInfo(idx,
        Model,
        EPROMVersion,
        IOResource,
        &DevVersion,
        &Type,
        &dwCaps,
        &wChipProtocol,
        &dwMaxCards);

        if(ret)
            //error - see Generic Error Codes
        else
            //success
```

4.5.IDC_GetDeviceStatus

int IDC_GetDeviceStatus(int idx, int *status);

Get the general device status.

Parameters

Input:

int idx: Same index returned by IDC_Open.

Output:

*int *status*: Returns device status. Possible values:

```
0x00000001 - Offline
0x00000002 - PowerOff
0x00000004 - Busy
0x00000008 – Communication Error
0x00000010 – Hardware Error
0x00000020 – Traction error
0x00000040 – Shutter failure
0x00000080 - Jam Error
```

Example Code:

```
int ret = 0;
int status = 0;

ret = IDC_GetDeviceStatus(idx, &status);
```

```
if(ret)
    //error - see Generic Error Codes or status errors
else
{
    //success reading status
    if(status)
        // Device has some error
}
```

4.6.IDC_GetMediaStatus

int IDC_GetMediaStatus(int idx, int *status);

Get card position.

Parameters

Input:

int idx: Same index returned by IDC_Open.

Output:

*int *status*: Returns device status. Possible values:

0x00000001 – No card available.

0x00000002 – Card detected at front position

0x00000004 – Card is completely inserted

0x00000008 – Card jammed

0x00000010 – Card position status not supported

0x00000020 – Card position unknown

0x00000040 – Card is locked by shutter

0x00008000 – Card detected at rear sensor but not at front sensor. It may indicate a front sensor failure

Example Code:

```
int ret = 0;
int status = 0;

ret = IDC_GetMediaStatus (idx, &status);
if(ret)
    //error - see Generic Error Codes
else
    //success
    if(status & 0x04)
        // Card completely inserted
```

4.7.IDC_GetRetainStatus

int IDC_GetRetainStatus(int idx, int *status);

Not implemented.

4.8.IDC_Enable

int IDC_Enable(int idx, int type);

This function enable magnetic and chip operations and must be called before calling IDC_ReadTracks and IDC_ChipIO functions. For magnetic operations, the device's buffer is cleared.

Parameters

Input:

int idx: Same index returned by IDC_Open.

int type: Type of operation. Possible values:

0x00000001 – Enable magnetic operations

0x00000002 – Enable smart chip operations

Example:

```
int ret = 0;

ret = IDC_Enable(idx, 0x01 | 0x02); // Enable magnetic and smart chip cards
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.9.IDC_CancelEnable

`int IDC_CancelEnable(int idx, int type);`

Disable magnetic and chip operations.

Parameters

Input:

int idx: Same index returned by IDC_Open.

int type: Type of operation. Possible values:

0x00000001 – Disable magnetic operations

0x00000002 – Disable smart chip operations

Example:

```
int ret = 0;

ret = IDC_CancelEnable(idx, 0x01 | 0x02); // Disable magnetic and smart chip cards
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.10. IDC_ReadTracks

```
int IDC_ReadTracks(int idx, int tracks,
                           char *track1, int *track1_size,
                           char *track2, int *track2_size,
                           char *track3, int *track3_size);
```

This function reads the magnetic tracks from card as a null terminated string. It's possible read any track in the same call passing the required track through parameter. Before reading, the IDC_Enable function must be called to enable magnetic reading.

Parameters

Input:

int idx: Same index returned by IDC_Open.

int tracks: Indicate which track will be read. All the tracks can be read in the same call. Possible values:

0x00000001 – Read magnetic track 1.

0x00000002 – Read magnetic track 2.

0x00000004 – Read magnetic track 3.

Output:

*char *track1*: Pointer to null terminated string read from track 1

*int *track1 size*: String length of track 1

*char *track1*: Pointer to null terminated string read from track 2

*int *track1 size*: String length of track 2

*char *track1*: Pointer to null terminated string read from track 3

*int *track1 size*: String length of track 3

Example:

```
int ret = 0;
char track_string1[512], track_string2[512], track_string3[512];
int track_size1, track_size2, track_size3;

ret = IDC_Enable(idx, 0x01); // Enable magnetic
if(ret)
    //error - see Generic Error Codes
else
{
    ret = IDC_ReadTracks(idx, 0x01 | 0x02 | 0x04, // Read track 1, 2 and 3
                        track_string1, &track_size1,
                        track_string2, &track_size2,
                        track_string3, &track_size3);
    if(ret)
        //error - see Generic Error Codes
    else
        // Read tracks successful
}
```

4.11. IDC_WriteTracks

```
int IDC_WriteTracks(int idx, int tracks,
                    char *track1, int track1_size,
                    char *track2, int track2_size,
                    char *track3, int track3_size);
```

Not implemented.

4.12. IDC_Contact

int IDC_Contact(int idx, unsigned char chip_type, unsigned char *data, int *length);

This function locks the card, performs a chip contact and reads its ATR data. The card must be completely inserted in the reader before call this function. The card position can be monitored by IDC_GetMediaStatus function.

Parameters

Input:

int idx: Same index returned by IDC_Open.

unsigned char chip type: Select the chip type to perform a contact. Possible values are:

0x00 – Smart card

0x01 to 0x0n – Select the SAM module 1 to n.

Output:

*unsigned char *data*: Returns an array of bytes read from card

*int *length*: Length of data read from card.

Example:

```
int ret = 0;
unsigned char data[128];
int length;

ret = IDC_Contact(idx, 0x00, data, &length); // Select smart card
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.13. IDC_Release

int IDC_Release(int idx);

This function release the card locked by IDC_Contact function.

Parameters

Input:

int idx: Same index returned by IDC_Open.

Example:

```
int ret = 0;

ret = IDC_Release(idx);
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.14. IDC_ChipIO

```
int IDC_ChipIO(int idx, unsigned char chip_type,
               unsigned char *data_out, int length_out,
               unsigned char *data_in, int *length_in);
```

This function performs data exchange with chip.

Parameters

Input:

int idx: Same index returned by IDC_Open.

unsigned char chip type: Select the chip type to perform a contact. Possible values are:

0x00 – Smart card

0x01 to 0x0n – Select the SAM module 1 to n.

*int *length in*: Maximum length of data read.

Output:

*unsigned char *data out*: Pointer to data will be sent to chip

int length out: Length of the data sent to chip. Length 0 mean string data.

*unsigned char *data in*: Pointer to response read from chip.

*int *length in*: Length of data read from chip.

Example:

```
int ret = 0;
unsigned char data_out[1024];
unsigned char data_in[1024];
int length_in = 0;

memset(data_out, 0, sizeof(data_out));
memset(data_in, 0, sizeof(data_in));

// mount frame to request challenge key. Size = 5 bytes
data_out[0] = 0x00;
data_out[1] = 0x84;
data_out[2] = 0x00;
data_out[3] = 0x00;
data_out[5] = 0x08;

// Max data read
length_in = 512;

ret = IDC_ChipIO(idx, 0x00,
                data_out, 5,
                data_in, &length_in);
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.15. IDC_Capture

`int IDC_Capture(int idx);`

Not implemented.

4.16. IDC_Eject

`int IDC_Eject(int idx);`

Not implemented.

4.17. IDC_ResetCount

`int IDC_ResetCount(int idx);`

Not implemented.

4.18. IDC_Led

`int IDC_Led(int idx, int led, int state);`

This function sets the state of the leds available in the device. State and Led available depends on reader.

Parameters

Input:

int idx: Same index returned by IDC_Open.

int led: Select the led to set the state. Possible values:

```
0x00000001 – Led Green
0x00000002 – Led Red
0x00000004 – Led Orange
```

int state: Select the new state of the led selected in the *led* parameter.

```
0x00000001 – Led Off.
0x00000002 – Led On.
0x00000004 – Led Blink.
```

Example:

```c
int ret = 0;

ret = IDC_Led(idx, 0x01, 0x02); // Turn On (0x02) Led green (0x01)
if(ret)
    //error - see Generic Error Codes
else
    //success

ret = IDC_Led(idx, 0x02, 0x01); // Turn Off (0x01) Led red (0x02)
if(ret)
    //error - see Generic Error Codes
else
    //success
```

4.19. IDC_LastError

int IDC_LastError(void);

This function returns the last error returned by last function call.

Example:

```c
int ret = 0;

ret = IDC_LastError();
printf("Last Error: %d\n", ret);
```

5. Generic Error Codes

Error Code	Description
-10001	Max number of process reached
-10002	Invalid service index
-10003	Invalid memory pointer
-10004	Unexpected error in MGR Class
-10005	Customer defined not found
-10006	Device defined not found
-20001	Invalid service name, section not found in configuration file
-20002	Configuration not defined
-20003	Configuration file not found

-20004	Error in get module path
-20005	Unexpected error in CFG class
-20006	Invalid configuration value
-30001	Generic erro in SEM
-30002	Timeout of semaphore
-30003	Error creating semaphore
-30004	Error closing semaphore
-30005	Invalid semaphore
-30006	Semaphore release fail
-30007	Fail in ini sec descriptor of semaphore
-30008	Fail in set sec descriptor of semaphore
-40001	Port not opened
-40002	Invalid protocol
-40003	Open failed
-40004	Write failed
-40005	Read failed
-40006	CRC16 dont match
-40007	Invalid result pointer
-40008	Invalid size of packet or parameter
-40009	Invalid response
-40010	Access denied open port
-40011	Access denied open port
-40012	Communication timeout
-50001	Command not implemented
-50002	Invalid parameter
-50003	Command not supported
-50004	Invalid response
-60001	Invalid pointer in LOG class
-60002	Log Disabled
-70001	Invalid pointer INT class, pointer of parammeter
-80001	Out of memory, error in allocation
-80002	Generic unexpected error

6. Device specific Error Codes

Error Code	Description
1	Function Read track failed
2	Function Contact failed
3	Function Release Failed
4	Function ChipIO Failed
5	Function Led Failed